МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ

ОДЕССКИЙ НАЦИОНАЛЬНЫЙ МОРСКОЙ УНИВЕРСИТЕТ

Кафедра «Информационных технологий»

Отчет по лабораторной работе №5

Тема: «ДЕРЕВЬЯ ПРИНЯТИЯ РЕШЕНИЙ»

Выполнил:

студент 3к. 2гр. КСФ

Талабишка Р.Р.Проверил:

Рудниченко Н.Д.

Одесса 2017

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 2

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 3

ПРАКТИЧЕСКАЯ ЧАСТЬ 5

ВЫВОД 12

ВВЕДЕНИЕ

Деревья решений – это способ представления правил в иерархической, последовательной структуре, где каждому объекту соответствует единственный узел, дающий решение.

Основные понятиями из теории деревьев решений являются:

­ Объект – пример, шаблон, наблюдение

­ Атрибут – признак, независимая переменная, свойство

­ Метка класса – зависимая переменная, целевая переменная, признак определяющий класс объекта

­ Узел – внутренний узел дерева, узел проверки

­ Лист – конечный узел дерева, узел решения

­ Проверка – условие в узле

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Деревья решений – это способ представления правил в иерархической, последовательной структуре, где каждому объекту соответствует единственный узел, дающий решение.

Основные понятиями из теории деревьев решений являются:

­ Объект – пример, шаблон, наблюдение

­ Атрибут – признак, независимая переменная, свойство

­ Метка класса – зависимая переменная, целевая переменная, признак определяющий класс объекта

­ Узел – внутренний узел дерева, узел проверки

­ Лист – конечный узел дерева, узел решения

­ Проверка – условие в узле

Под правилом понимается логическая конструкция, представленная в виде "если ... то ...".

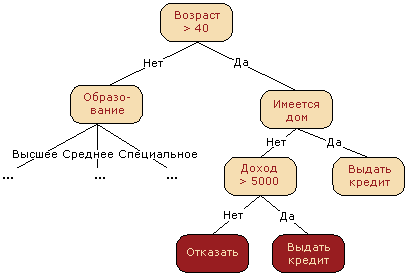


Рис. 1. Пример дерева принятия решений о выдаче кредита

Область применения деревья решений в настоящее время широка, но все задачи, решаемые этим аппаратом могут быть объединены в следующие три класса:

1. Описание данных: Деревья решений позволяют хранить информацию о данных в компактной форме, вместо них мы можем хранить дерево решений, которое содержит точное описание объектов.
2. Классификация: Деревья решений отлично справляются с задачами классификации, т.е. отнесения объектов к одному из заранее известных классов. Целевая переменная должна иметь дискретные значения.
3. Регрессия: Если целевая переменная имеет непрерывные значения, деревья решений позволяют установить зависимость целевой переменной от независимых(входных) переменных. Например, к этому классу относятся задачи численного прогнозирования(предсказания значений целевой переменной).
   1. Построение дерева принятия решений

Пусть задано некоторое обучающее множество T, содержащее объекты (примеры), каждый из которых характеризуется m атрибутами (атрибутами), причем один из них указывает на принадлежность объекта к определенному классу.

Пусть через {C1, C2, ... Ck} обозначены классы(значения метки класса), тогда существуют 3 ситуации:

1. Множество T содержит один или более примеров, относящихся к одному классу Ck. Тогда дерево решений для Т – это лист, определяющий класс Ck;
2. Множество T не содержит ни одного примера, т.е. пустое множество. Тогда это снова лист, и класс, ассоциированный с листом, выбирается из другого множества отличного от T, скажем, из множества, ассоциированного с родителем;
3. Множество T содержит примеры, относящиеся к разным классам. В этом случае следует разбить множество T на некоторые подмножества. Для этого выбирается один из признаков, имеющий два и более отличных друг от друга значений O1, O2, ... On. T разбивается на подмножества T1, T2, ... Tn, где каждое подмножество Ti содержит все примеры, имеющие значение Oi для выбранного признака. Это процедура будет рекурсивно продолжаться до тех пор, пока конечное множество не будет состоять из примеров, относящихся к одному и тому же классу.

Вышеописанная процедура лежит в основе многих современных алгоритмов построения деревьев решений, этот метод известен еще под названием разделения и захвата (divide and conquer). Очевидно, что при использовании данной методики, построение дерева решений будет происходит сверху вниз.

Поскольку все объекты были заранее отнесены к известным нам классам, такой процесс построения дерева решений называется обучением с учителем (supervised learning). Процесс обучения также называют индуктивным обучением или индукцией деревьев (tree induction).

На сегодняшний день существует значительное число алгоритмов, реализующих деревья решений CART, C4.5, NewId, ITrule, CHAID, CN2 и т.д. Но наибольшее распространение и популярность получили следующие два:

CART (Classification and Regression Tree) – это алгоритм построения бинарного дерева решений – дихотомической классификационной модели. Каждый узел дерева при разбиении имеет только двух потомков. Как видно из названия алгоритма, решает задачи классификации и регрессии.

C4.5 – алгоритм построения дерева решений, количество потомков у узла не ограничено. Не умеет работать с непрерывным целевым полем, поэтому решает только задачи классификации.

3.3 Критерии выбора атрибутов, остановка обучения и отсечение ветвей

Большинство из известных алгоритмов являются "жадными алгоритмами". Если один раз был выбран атрибут, и по нему было произведено разбиение на подмножества, то алгоритм не может вернуться назад и выбрать другой атрибут, который дал бы лучшее разбиение. И поэтому на этапе построения нельзя сказать даст ли выбранный атрибут, в конечном итоге, оптимальное разбиение.

При построении деревьев решений особое внимание уделяется следующим вопросам: выбору критерия атрибута, по которому пойдет разбиение, остановки обучения и отсечения ветвей. Рассмотрим все эти вопросы по порядку.

3.3.1 Правило разбиения. Каким образом следует выбрать признак?

Для построения дерева на каждом внутреннем узле необходимо найти такое условие (проверку), которое бы разбивало множество, ассоциированное с этим узлом на подмножества. В качестве такой проверки должен быть выбран один из атрибутов. Общее правило для выбора атрибута можно сформулировать следующим образом: выбранный атрибут должен разбить множество так, чтобы получаемые в итоге подмножества состояли из объектов, принадлежащих к одному классу, или были максимально приближены к этому, т.е. количество объектов из других классов ("примесей") в каждом из этих множеств было как можно меньше.

Были разработаны различные критерии, рассмотрим два из них:

1. Теоретико-информационный критерий

Алгоритм C4.5, усовершенствованная версия алгоритма ID3 (Iterative Dichotomizer), использует теоретико-информационный подход. Для выбора наиболее подходящего атрибута, предлагается следующий критерий:

f1

где, Info(T) – энтропия множества T, а

f2

Множества T1, T2, ... Tn получены при разбиении исходного множества T по проверке X. Выбирается атрибут, дающий максимальное значение.

1. Статистический критерий

Алгоритм CART использует так называемый индекс Gini (в честь итальянского экономиста Corrado Gini), который оценивает "расстояние" между распределениями классов.

gini

где c – текущий узел, а pj – вероятность класса j в узле c.

3.3.2 Правило остановки. Разбивать дальше узел или отметить его как лист?

В дополнение к основному методу построения деревьев решений были предложены следующие правила:

­ Использование статистических методов для оценки целесообразности дальнейшего разбиения, так называемая "ранняя остановка" (prepruning). В конечном счете "ранняя остановка" процесса построения привлекательна в плане экономии времени обучения, но здесь уместно сделать одно важное предостережение: этот подход строит менее точные классификационные модели и поэтому ранняя остановка крайне нежелательна.

­ Ограничить глубину дерева. Остановить дальнейшее построение, если разбиение ведет к дереву с глубиной превышающей заданное значение.

­ Разбиение должно быть нетривиальным, т.е. получившиеся в результате узлы должны содержать не менее заданного количества примеров.

Этот список эвристических правил можно продолжить, но на сегодняшний день не существует такого, которое бы имело большую практическую ценность. К этому вопросу следует подходить осторожно, так как многие из них применимы в каких-то частных случаях.

3.3.3 Правило отсечения. Каким образом ветви дерева должны отсекаться?

Очень часто алгоритмы построения деревьев решений дают сложные деревья, которые "переполнены данными", имеют много узлов и ветвей. Такие "ветвистые" деревья очень трудно понять. К тому же ветвистое дерево, имеющее много узлов, разбивает обучающее множество на все большее количество подмножеств, состоящих из все меньшего количества объектов.

Ценность правила, справедливого скажем для 2-3 объектов, крайне низка, и в целях анализа данных такое правило практически непригодно. Гораздо предпочтительнее иметь дерево, состоящее из малого количества узлов, которым бы соответствовало большое количество объектов из обучающей выборки. И тут возникает вопрос: а не построить ли все возможные варианты деревьев, соответствующие обучающему множеству, и из них выбрать дерево с наименьшей глубиной? К сожалению, это задача является NP-полной.

Для решения вышеописанной проблемы часто применяется так называемое отсечение ветвей (pruning).

Пусть под точностью (распознавания) дерева решений понимается отношение правильно классифицированных объектов при обучении к общему количеству объектов из обучающего множества, а под ошибкой – количество неправильно классифицированных. Предположим, что нам известен способ оценки ошибки дерева, ветвей и листьев. Тогда, возможно использовать следующее простое правило:

­ построить дерево;

­ отсечь или заменить поддеревом те ветви, которые не приведут к возрастанию ошибки.

В отличии от процесса построения, отсечение ветвей происходит снизу вверх, двигаясь с листьев дерева, отмечая узлы как листья, либо заменяя их поддеревом.

Хотя отсечение не является панацеей, но в большинстве практических задач дает хорошие результаты, что позволяет говорить о правомерности использования подобной методики.

3.4 Преимущества использования деревьев решений

Рассмотрев основные проблемы, возникающие при построении деревьев, было бы несправедливо не упомянуть об их достоинствах:

­ быстрый процесс обучения;

­ генерация правил в областях, где эксперту трудно формализовать свои знания;

­ извлечение правил на естественном языке;

­ интуитивно понятная классификационная модель;

­ высокая точность прогноза, сопоставимая с другими методами (статистика, нейронные сети);

­ построение непараметрических моделей.

В силу этих и многих других причин, методология деревьев решений является важным инструментом в работе каждого специалиста, занимающегося анализом данных, вне зависимости от того практик он или теоретик.

Деревья решений являются прекрасным инструментом в системах поддержки принятия решений, интеллектуального анализа данных (data mining).

ПРАКТИЧЕСКАЯ ЧАСТЬ

package tpr5;

import java.awt.\*;

import java.awt.geom.\*;

import java.awt.event.\*;

import java.io.\*;

import javax.swing.\*;

import javax.swing.event.\*;

import java.awt.image.\*;

import javax.imageio.\*;

import javax.swing.filechooser.FileFilter;

public class ImageEdit

{

// Режим рисования

int rezhim=0;

int xPad;

int xf;

int yf;

int yPad;

int thickness;

boolean pressed=false;

// текущий цвет

Color maincolor;

MyFrame f;

MyPanel japan;

JButton colorbutton;

JColorChooser tcc;

// поверхность рисования

BufferedImage imag;

// если мы загружаем картинку

boolean loading=false;

String fileName;

public ImageEdit()

{

f=new MyFrame("Графический редактор");

f.setSize(350,350);

f.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

maincolor=Color.black;

JMenuBar menuBar = new JMenuBar();

f.setJMenuBar(menuBar);

menuBar.setBounds(0,0,350,30);

JMenu fileMenu = new JMenu("Файл");

menuBar.add(fileMenu);

Action loadAction = new AbstractAction("Загрузить")

{

public void actionPerformed(ActionEvent event)

{

JFileChooser jf= new JFileChooser();

int result = jf.showOpenDialog(null);

if(result==JFileChooser.APPROVE\_OPTION)

{

try

{

// при выборе изображения подстраиваем размеры формы

// и панели под размеры данного изображения

fileName = jf.getSelectedFile().getAbsolutePath();

File iF= new File(fileName);

jf.addChoosableFileFilter(new TextFileFilter(".png"));

jf.addChoosableFileFilter(new TextFileFilter(".jpg"));

imag = ImageIO.read(iF);

loading=true;

f.setSize(imag.getWidth()+40, imag.getWidth()+80);

japan.setSize(imag.getWidth(), imag.getWidth());

japan.repaint();

} catch (FileNotFoundException ex) {

JOptionPane.showMessageDialog(f, "Такого файла не существует");

}

catch (IOException ex) {

JOptionPane.showMessageDialog(f, "Исключение ввода-вывода");

}

catch (Exception ex) {

}

}

}

};

JMenuItem loadMenu = new JMenuItem(loadAction);

fileMenu.add(loadMenu);

Action saveAction = new AbstractAction("Сохранить")

{

public void actionPerformed(ActionEvent event)

{

try

{

JFileChooser jf= new JFileChooser();

// Создаем фильтры файлов

TextFileFilter pngFilter = new TextFileFilter(".png");

TextFileFilter jpgFilter = new TextFileFilter(".jpg");

if(fileName==null)

{

// Добавляем фильтры

jf.addChoosableFileFilter(pngFilter);

jf.addChoosableFileFilter(jpgFilter);

int result = jf.showSaveDialog(null);

if(result==JFileChooser.APPROVE\_OPTION)

{

fileName = jf.getSelectedFile().getAbsolutePath();

}

}

// Смотрим какой фильтр выбран

if(jf.getFileFilter()==pngFilter)

{

ImageIO.write(imag, "png", new File(fileName+".png"));

}

else

{

ImageIO.write(imag, "jpeg", new File(fileName+".jpg"));

}

}

catch(IOException ex)

{

JOptionPane.showMessageDialog(f, "Ошибка ввода-вывода");

}

}

};

JMenuItem saveMenu = new JMenuItem(saveAction);

fileMenu.add(saveMenu);

Action saveasAction = new AbstractAction("Сохранить как...")

{

public void actionPerformed(ActionEvent event)

{

try

{

JFileChooser jf= new JFileChooser();

// Создаем фильтры для файлов

TextFileFilter pngFilter = new TextFileFilter(".png");

TextFileFilter jpgFilter = new TextFileFilter(".jpg");

// Добавляем фильтры

jf.addChoosableFileFilter(pngFilter);

jf.addChoosableFileFilter(jpgFilter);

int result = jf.showSaveDialog(null);

if(result==JFileChooser.APPROVE\_OPTION)

{

fileName = jf.getSelectedFile().getAbsolutePath();

}

// Смотрим какой фильтр выбран

if(jf.getFileFilter()==pngFilter)

{

ImageIO.write(imag, "png", new File(fileName+".png"));

}

else

{

ImageIO.write(imag, "jpeg", new File(fileName+".jpg"));

}

}

catch(IOException ex)

{

JOptionPane.showMessageDialog(f, "Ошибка ввода-вывода");

}

}

};

JMenuItem saveasMenu = new JMenuItem(saveasAction);

fileMenu.add(saveasMenu);

japan = new MyPanel();

japan.setBounds(30,30,260,260);

japan.setBackground(Color.white);

japan.setOpaque(true);

f.add(japan);

JToolBar toolbar = new JToolBar("Toolbar", JToolBar.VERTICAL);

JButton penbutton = new JButton(new ImageIcon("pen.png"));

penbutton.addActionListener(new ActionListener()

{

public void actionPerformed(ActionEvent event)

{

rezhim=0;

}

});

toolbar.add(penbutton);

/\* JButton brushbutton = new JButton(new ImageIcon("brush.png"));

brushbutton.addActionListener(new ActionListener()

{

public void actionPerformed(ActionEvent event)

{

rezhim=1;

}

});

toolbar.add(brushbutton);\*/

JButton lasticbutton = new JButton(new ImageIcon("lastic.png"));

lasticbutton.addActionListener(new ActionListener()

{

public void actionPerformed(ActionEvent event)

{

rezhim=2;

}

});

toolbar.add(lasticbutton);

JButton textbutton = new JButton(new ImageIcon("text.png"));

textbutton.addActionListener(new ActionListener()

{

public void actionPerformed(ActionEvent event)

{

rezhim=3;

}

});

toolbar.add(textbutton);

JButton linebutton = new JButton(new ImageIcon("line.png"));

linebutton.addActionListener(new ActionListener()

{

public void actionPerformed(ActionEvent event)

{

rezhim=4;

}

});

toolbar.add(linebutton);

JButton elipsbutton = new JButton(new ImageIcon("elips.png"));

elipsbutton.addActionListener(new ActionListener(){

public void actionPerformed(ActionEvent event)

{

rezhim=5;

}

});

toolbar.add(elipsbutton);

toolbar.setBounds(0, 0, 30, 300);

f.add(toolbar);

tcc = new JColorChooser(maincolor);

tcc.getSelectionModel().addChangeListener(new ChangeListener()

{

public void stateChanged(ChangeEvent e)

{

maincolor = tcc.getColor();

colorbutton.setBackground(maincolor);

}

});

japan.addMouseMotionListener(new MouseMotionAdapter()

{

public void mouseDragged(MouseEvent e)

{

if (pressed==true)

{

Graphics g = imag.getGraphics();

Graphics2D g2 = (Graphics2D)g;

// установка цвета

g2.setColor(maincolor);

switch (rezhim)

{

// карандаш

case 0:

g2.drawLine(xPad, yPad, e.getX(), e.getY());

break;

// кисть

case 1:

g2.setStroke(new BasicStroke(3.0f));

g2.drawLine(xPad, yPad, e.getX(), e.getY());

break;

// ластик

case 2:

g2.setStroke(new BasicStroke(3.0f));

g2.setColor(Color.WHITE);

g2.drawLine(xPad, yPad, e.getX(), e.getY());

break;

}

xPad=e.getX();

yPad=e.getY();

}

japan.repaint();

}

});

japan.addMouseListener(new MouseAdapter()

{

public void mouseClicked(MouseEvent e) {

Graphics g = imag.getGraphics();

Graphics2D g2 = (Graphics2D)g;

// установка цвета

g2.setColor(maincolor);

switch (rezhim)

{

// карандаш

case 0:

g2.drawLine(xPad, yPad, xPad+1, yPad+1);

break;

// кисть

case 1:

g2.setStroke(new BasicStroke(3.0f));

g2.drawLine(xPad, yPad, xPad+1, yPad+1);

break;

// ластик

case 2:

g2.setStroke(new BasicStroke(3.0f));

g2.setColor(Color.WHITE);

g2.drawLine(xPad, yPad, xPad+1, yPad+1);

break;

// текст

case 3:

// устанавливаем фокус для панели,

// чтобы печатать на ней текст

japan.requestFocus();

break;

}

xPad=e.getX();

yPad=e.getY();

pressed=true;

japan.repaint();

}

public void mousePressed(MouseEvent e) {

xPad=e.getX();

yPad=e.getY();

xf=e.getX();

yf=e.getY();

pressed=true;

}

public void mouseReleased(MouseEvent e) {

Graphics g = imag.getGraphics();

Graphics2D g2 = (Graphics2D)g;

// установка цвета

g2.setColor(maincolor);

// Общие рассчеты для овала и прямоугольника

int x1=xf, x2=xPad, y1=yf, y2=yPad;

if(xf>xPad)

{

x2=xf; x1=xPad;

}

if(yf>yPad)

{

y2=yf; y1=yPad;

}

switch(rezhim)

{

// линия

case 4:

g.drawLine(xf, yf, e.getX(), e.getY());

break;

// круг

case 5:

g.drawOval(x1, y1, (x2-x1), (y2-y1));

break;

// прямоугольник

case 6:

g.drawRect(x1, y1, (x2-x1), (y2-y1));

break;

}

xf=0; yf=0;

pressed=false;

japan.repaint();

}

});

japan.addKeyListener(new KeyAdapter()

{

public void keyReleased(KeyEvent e)

{

// устанавливаем фокус для панели,

// чтобы печатать на ней текст

japan.requestFocus();

}

public void keyTyped(KeyEvent e)

{

if(rezhim==3){

Graphics g = imag.getGraphics();

Graphics2D g2 = (Graphics2D)g;

// установка цвета

g2.setColor(maincolor);

g2.setStroke(new BasicStroke(2.0f));

String str = new String("");

str+=e.getKeyChar();

g2.setFont(new Font("Arial", 0, 15));

g2.drawString(str, xPad, yPad);

xPad+=10;

// устанавливаем фокус для панели,

// чтобы печатать на ней текст

japan.requestFocus();

japan.repaint();

}

}

});

f.addComponentListener(new ComponentAdapter() {

public void componentResized(java.awt.event.ComponentEvent evt) {

// если делаем загрузку, то изменение размеров формы

// отрабатываем в коде загрузки

if(loading==false)

{

japan.setSize(f.getWidth()-40, f.getHeight()-80);

BufferedImage tempImage = new BufferedImage(japan.getWidth(), japan.getHeight(), BufferedImage.TYPE\_INT\_RGB);

Graphics2D d2 = (Graphics2D) tempImage.createGraphics();

d2.setColor(Color.white);

d2.fillRect(0, 0, japan.getWidth(), japan.getHeight());

tempImage.setData(imag.getRaster());

imag=tempImage;

japan.repaint();

}

loading=false;

}

});

f.setLayout(null);

f.setVisible(true);

}

public static void main(String[] args) {

SwingUtilities.invokeLater(new Runnable() {

public void run() {

new ImageEdit();

}

});

}

class ColorDialog extends JDialog

{

public ColorDialog(JFrame owner, String title)

{

super(owner, title, true);

add(tcc);

setSize(200, 200);

}

}

class MyFrame extends JFrame

{

public void paint(Graphics g)

{

super.paint(g);

}

public MyFrame(String title)

{

super(title);

}

}

class MyPanel extends JPanel

{

public MyPanel()

{ }

public void paintComponent (Graphics g)

{

if(imag==null)

{

imag = new BufferedImage(this.getWidth(), this.getHeight(), BufferedImage.TYPE\_INT\_RGB);

Graphics2D d2 = (Graphics2D) imag.createGraphics();

d2.setColor(Color.white);

d2.fillRect(0, 0, this.getWidth(), this.getHeight());

}

super.paintComponent(g);

g.drawImage(imag, 0, 0,this);

}

}

// Фильтр картинок

class TextFileFilter extends FileFilter

{

private String ext;

public TextFileFilter(String ext)

{

this.ext=ext;

}

public boolean accept(java.io.File file)

{

if (file.isDirectory()) return true;

return (file.getName().endsWith(ext));

}

public String getDescription()

{

return "\*"+ext;

}

}

}

Результат выполнения программы представлен на рисунке 1.

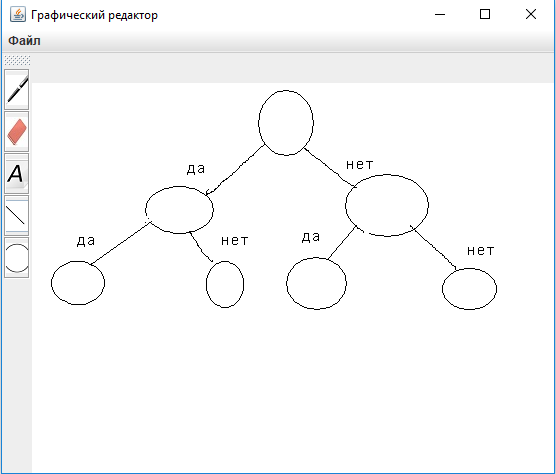


Рисунок 1

ВЫВОД

Деревья решений – это способ представления правил в иерархической, последовательной структуре, где каждому объекту соответствует единственный узел, дающий решение.

Этот метод является отличным для шаблонных способов принятия решения, и используется во многих структурах, так как предоставляет иерархическое сценарное развитие событий.